(2)

FINAL REPORT
for

RECOVERY TECHNIQUES FOR REAL TIME ELECTRONIC SYSTEMS

for the period

MARCH 1, 1988 through DECEMBER 31, 1991

Contract # N00014-89-K-0089

Submitted to

The Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217-5000

Attention: Dr. Clifford Lau
Electronics Division, Code 1114

March 30, 1992

Jacob A. Abraham, Principal Investigator
Computer Engineering Research Center
Bureau of Engineering Research
The University of Texas at Austin
2201 Donley Drive, Suite 395
Austin, Texas 78758

Introduction

The objective of this research was to develop new analytical techniques for evaluating the effectiveness of recovery in real time systems, and to design novel recovery techniques specifically tailored to real time applications. We have developed a variety of techniques described in more detail below.

Design of Error Detection and Recovery Techniques

This section describes techniques for the design of digital systems which will support on-line error detection and recovery with low performance overhead. Results have been obtained in the areas of encoding techniques, asynchronous recovery, and robust data structures.

Real-Number Codes For Fault-Tolerant Matrix Operations On Processor Arrays

Various checksum codes have been suggested for fault-tolerant matrix computations on processor arrays. Use of these codes is limited due to inflexibility of the encoding schemes and also due to potential numerical problems. Numerical errors may also be construed as errors due to physical faults in the system. We have developed a generalization of the existing schemes as a possible solution to these shortcomings.

We have proved that linearity is a necessary and sufficient condition for every code that can be used for fault-tolerant matrix operations such as matrix transposition, addition, multiplication, and matrix-vector multiplication. We also proved that for every linear finite-field code, there exists a real-number code having similar error diagnosing capabilities as the finite-fields code. Numerical problems associated with high-level encoding such as real-number encoding were addressed and solutions have been proposed. We observed that by selecting a particular code for a specific application, we can minimize the numerical errors in the check elements. The objective of selecting a particular code should be to minimize the error reflectivity ratio rather than minimizing the error.

We conducted a number of experiments to find the maximum numerical error produced during matrix multiplication using various codes. A rule of thumb for selecting a particular code has been formulated on the basis of experimental results. When roundoff is the larger problem, one should use normalized encoder vectors. The weight of the normalized encoder vector element may lie close to the inverse of the average of the data elements. For overflow problems, one should use average checksum codes or codes reproduced by periodic encoder vectors. Periodic encoder vectors are especially suitable when the majority of the data elements have the same sign and are randomly distributed in magnitude.

The hardware overhead associated with the general encoding schemes, in terms of the number of processors, is the same as that of the existing schemes except that each processing unit is slightly more complicated due to additional registers required for holding the temporary results. A simulation environment was used to evaluate the performance overhead of the fault tolerance schemes. The time overheads observed from these experiments conform to the theoretically predicted results.

Asynchronous Recovery using Message Logs

Many checkpointing strategies have been proposed for use in distributed computing systems, but these may not be suitable for real-time systems because of their severe performance degradation under high error rates. One such approach is based on centralized published communications where checkpoint information is maintained for all processes, as well as a record of all messages sent to each process and a count of all messages sent by each process. We have developed an extension to this model using message logs and local asynchronous checkpoints. We developed an analysis technique which showed that a system using this method will perform better under high error rates.

We also developed a simulation in order to compare globally coordinated checkpointing strategies with our asynchronous strategy. The results showed that the asynchronous strategy offered less performance degradation as the failure rates increased. This leads to the conclusion that a designer of real-time distributed system should consider asynchronous checkpointing as a way of ensuring adequate performance under worst-case conditions.

A Modular Robust Binary Tree

Since computer systems utilize data structures to represent and manage their data, concurrent error detection and correction techniques include the use of robust data structures in systems requiring high reliability. Many techniques have been proposed for adding redundancy to data structures in order to detect and correct errors in their structural components.

We have designed and implemented a new robust binary tree architecture, called the Modular Robust Binary (MRB) tree. The MRB tree is based on a modular approach for the construction of the robust storage structure. Basic units, or modules, are interconnected in a systematic way to form the desired binary tree. The basic module is composed of three nodes, a parent node, a left child node, and a right child node with two disjoint paths between each pair of the nodes. Single errors can be corrected in constant time; during a forward move from the root to the leaves, tree pointers are used to traverse down the tree, whereas during a backward move, a parent pointer is used to traverse towards the root of the tree.

Relationships between the three nodes are developed to establish that the proposed design is 2-detectable or 1-correctable at every module. Logical rules were designed to relate three attributes among the set of selected attributes utilized in the construction of the robust tree. These attributes are: a left-child pointer, a right-child pointer, a parent pointer, a sibling pointer, an identifier, and a *key*. The design of the MRB tree has included provisions for detecting two compensating errors. A *rules hypergraph* is consequently obtained which relates the set of attributes in a module in order to show the correctability of the MRB tree. A rules hypergraph is a hypergraph $G(V,E)$ where V is a set of attributes represented by the vertices in the hypergraph, and E is the set of rules represented by the hyperedges connecting these attributes.

In order to experimentally evaluate the properties and overhead of the proposed tree, an implementation of the robust data structure was coded in C++. Assertions utilizing the redundant structural information were placed at critical points in all the data structure routines (methods), including Search, Traverse, Insert, Delete and Find. If any of these assertions fails, a correction routine is invoked to locate the error as well as to apply cor-

rection procedures. A syndrome table (analogous to a hardware maintenance dictionary) is constructed in order to isolate the distinct errors.

Over 60,000 fault injection experiments were conducted using FERRARI (Fault and ERRor Automatic Real-time Injector), a software implemented fault injector, to measure the the effectiveness of the robust tree in contributing to the overall error detection capabilities of a system. The faults injected through software techniques emulated hardware faults, and would be detected only indirectly when the errors they caused resulted in damage to thr robust data structure. These experiments were conducted on a SUN4 workstation.

The experiments showed that many errors due to injected faults were detected by the SUN4 built-in error detection mechanisms such as illegal instruction, segmentation fault, bus error, and bad system calls. The redundancy in the MRB tree enabled the detection of errors due to 46.4% of the remaining faults. The performance overhead measured during a traversal of the tree was 160% compared with the reported overhead in the literature of 450% for a search routine. The overhead for construction of the tree which combines two routines, insert and search for a position, was 95%. The overhead for deletion of elements of the tree, which combines two routines, finding the selected element and deleting the element, was found to be 90%.

Accurate Evaluation Techniques for Large Systems

A model was developed for the analytical evaluation of large fault-tolerant systems. Techniques for incorporating probabilities into the model and methods for analyzing hierarchical systems were also developed. A fundamental technique of *functional partitioning* was developed for the analysis of large functions, and was applied to verifying the correctness of three different classes of functions which were hitherto intractable. A powerful software-based fault injection system, which accurately emulates hardware faults, was developed to evaluate proposed fault-tolerant systems.

A Model For The Analysis Of Fault-Tolerant Signal Processing Architectures

We developed a new model, using matrices, for the analysis of fault-tolerant multiprocessor systems. The relationship between processors computing useful data, the output data, and the check processors is defined in terms of matrix entries. Unlike the matrix based models proposed previously for the analysis of digital systems, this model uses only numerical computations rather than logical operations for the analysis of a system. We present algorithms to evaluate the fault detection and location capability of the system. These algorithms are much less complex than the existing ones. We also used the new model to analyze some fault-tolerant architectures proposed for signal processing applications.

In the conventional analysis of fault-tolerant multiprocessor systems, one assumes the existence of complete tests which always fail when there is a faulty processor. A processor can test another, and if the tested unit is faulty and the tester is fault-free, then the test is guaranteed to fail. However, in many sophisticated fault-tolerant systems a particular fault pattern can produce a number of different error patterns. The checking operations detect the errors directly and the faults indirectly. Therefore, fault analysis in such systems is much more complex than conventional fault analysis. The new model is specifically developed for

analyzing faults in systems using algorithm-based fault tolerance and similar techniques; however, application of the model for fault analysis of conventional redundancy techniques (such as duplication or triplication) is trivial.

We have defined three matrices, the PD (Processor-Data) matrix, the DC (Data-Check) matrix and the PC (Processor-Check) matrix, which we use to describe the system and its dependability characteristics. We have formulated algorithms for the analysis of systems for fault detectability and locatability. Unlike existing algorithms, the new algorithms do not need an exhaustive enumeration of faults and errors. This is achieved through an error collapsing technique using the proposed model. The model can also be used for the design of fault-tolerant systems. The model has been applied to the analysis of some fault-tolerant signal processing architectures.

Accurate Characterization Of Error Propagation In A Highly Parallel Architecture

Errors caused by faults in a system manifest themselves in the output results in various patterns. These output patterns of erroneous data, the nature of their propagation through the various modules of the system and their impact on the system are collectively referred to as *error characteristics*. We dealt with efficient methods of obtaining these error characteristics. Both experimental and analytical approaches to the problem were considered and a probabilistic approach to this problem was proposed. The effectiveness of this approach with respect to the accuracy of results and amount of computation involved was discussed.

In our experiments, we have used a powerful fault simulator called CHAMP - Concurrent Hierarchical and Multilevel Program for Simulation of VLSI Circuits. CHAMP is a switch level simulator that uses a hierarchical system description to avoid the huge memory requirements. It allows fault injection both at transistor and gate levels and allows mixed mode simulation wherein parts of the system can be simulated faster at a behavioral level by supplying a high level software description.

Although fault simulation provides a faster alternative to serial simulation or physical fault injection, it still is an extremely computationally intensive operation. Mixed mode simulation in CHAMP provides a remarkable speed advantage, but for complex systems this process still remains very expensive.

The overall model for probabilistic error characterization consists of two submodels. These are the error generation and the error propagation models.

This error generation model basically attempts to study the transformation of a fault into an error. This process is however very highly technology dependent. It is dependent upon the exact device level implementation of the system and the fault model considered. Therefore, it is not possible to generalize it for all systems. We proposed the use of fault simulation on the faulty module to generate the error pattern at a module output, which can then be propagated through the rest of the system using probabilistic error propagation.

Error propagation, unlike error generation, is more predictable on the basis of the functional behavior of the system and therefore its model is independent of the exact implementation and will find greater applicability. A new probabilistic technique has been proposed to model error propagation through the system. This approach uses the data flow graph of the system and the actual signal probabilities of each line to propagate the error probabilities

throughout the system. The next section deals with the development of this model.

Once the propagation probabilities are computed as above, error propagation at the system level is done by simulating the data flow graph of the system, wherein instead of actual data, we propagate the probability error on the line. Each of the nodes in the data flow graph represents a module in the system and the arcs connecting them represent the data flow between the modules. Thus, in the simulation of the data flow graph, each node uses the propagation probabilities within it, along with the input error probabilities, to compute the output error probabilities. These values are then propagated through the rest of the system.

The results show that the error probability on the output lines as predicted by the simulation of the probabilistic model was always within 15 percent of the actual fault simulation experiment. In addition, the model required an order of magnitude less computation time than simulation for 200 error patterns, and the difference in time between the two methods will increase as the number of error patterns increases.

Probabilistic Evaluation of On-Line Checks in Fault-Tolerant Multiprocessor Systems

The analysis of fault-tolerant multiprocessor systems that use concurrent error detection (CED) schemes is much more difficult than the analysis of conventional fault-tolerant architectures. Various analytical techniques have been proposed to evaluate CED schemes deterministically. However, these approaches are based on worst-case assumptions related to the failure of system components. Often, the evaluation results do not reflect the actual fault tolerance capabilities of a system.

We developed a probabilistic approach to evaluate on-line checks performed on fault-tolerant multiprocessor systems. In the probabilistic model, a probability is assigned to every output data element expressing the likelihood that the particular data element is erroneous. Further, probabilities are assigned to the checking operations to reflect their ability to detect various error patterns.

The various probabilities are derived based on a functional fault model and algorithms were developed for determining the probability of detection and location of faults by checking operations. The development of the algorithms was made simple by using the various probabilities in the framework of the matrix model we proposed previously. The techniques were applied to a variety of example systems. The analytical results showed that the probabilistic approach can handle situations that the deterministic model cannot, and can reduce the complexity of the computation without significantly sacrificing the accuracy of the results.

Hierarchical Design and Analysis of Fault-Tolerant Multiprocessor Systems Using Concurrent Error Detection

Large fault-tolerant multiprocessor systems are usually designed to handle error detection, fault isolation and recovery in a hierarchical fashion. However, existing techniques for analyzing such systems treat them at a single, flat level. We have developed hierarchical analysis techniques for such systems which require much less computational effort. The fault tolerance of the system at different levels of hierarchy was determined and the overall fault

tolerance was estimated from those values. We developed a composition technique to build large fault tolerant systems hierarchically using the concept of checks at different levels in the hierarchy. A small system of known fault detectability and locatability was replicated several times and new checks were added at the next higher level. Such checks at different levels can be introduced into most of the existing multiprocessor systems. We also developed an analysis technique based on our matrix model. Relationships between the fault detectability and locatability of a basic system were derived, and the corresponding values of the complete system were computed hierarchically. Finally, we extended the techniques to complex systems where individual processors produce multiple sets of data elements.

Functional Partitioning For Verification And Related Problems

We have developed *functional partitioning* methods to reduce the computational resources used in analyzing Boolean functions. One general partitioning strategy, *orthogonal partitioning*, divides a function's truth table into disjointed partitions; the Boolean and of any two subfunctions is always zero. Functional partitioning techniques can significantly augment current verification methods based on Ordered Binary Decision Diagrams (OBDDs). The design and specifications are partitioned into smaller functions that can be independently verified in a pairwise manner; these functions can even be ordered separately. This allows verification in polynomial space and time of several functions whose OBDD representations have been proven to be exponential in the number of output variables such as Bryant's hidden weighted bit function, as well as much more efficient verification of other difficult functions such as multipliers. Functional partitioning improves the efficiency of probabilistic verification as well, where even further improvements can be realized for the above functions. In the presence of design errors, these partitioning techniques can provide even greater efficiency gains.

We have shown how functionally partitioning a Boolean circuit can be used to enhance the efficiency of probabilistic and deterministic verification. For functions such as the *Fortune-Hopcroft-Schmidt* function or Bryant's *hidden-weighted-bit* function, verification in polynomial time and space is possible. Such functions were intractable using monolithic OBDD based methods. Similarly we have shown how such partitioning can easily reduce the space required for verifying a 16-bit multiplier by as much as three orders of magnitude.

Since the techniques allow corresponding pairs of partitions to be verified independently, functional partitioning is even more efficient in the presence of design errors, as a discrepancy between design and specification can be expected to appear in any given partition with reasonably high probability. For example, design errors were randomly introduced in c6288. The errors were introduced by simply inverting the functionality of a gate after every 100 lines in the circuit file; in each run one gate was erroneously set. Partitioned into 1024 orthogonal circuits, the inequivalence was verified by both probabilistic and deterministic methods in less than 20 seconds. Similar results were obtained for checking inequivalence in c3540 and the other difficult ISCAS circuits as well.

Functional partitioning leads to extremely efficient parallel algorithms. Partioned into 1024 subfunctions, it can also help verify the benchmark integer multiplier, C6288, where the largest graph requires no more than 1587 nodes, with the total intermediate space of any partition less than 18,000 nodes. On a Sun 4/280, no partition required more than one

CPU minute to be verified. For difficult functions, parallel verification can offer a speedup linear in the number of partitions.

Efficient Verification Of Multiplier And Other Difficult Functions Using IBDDs

It is well known that any Ordered Binary Decision Diagram (OBDD) representing a multiplier (or some other difficult Boolean functions) requires exponential space. Alternate representation schemes have been suggested for multipliers. However, such representations have difficulties in capturing even some simple modifications in a multiplier circuit. We have developed Indexed BDDs (IBDDs), a generally applicable representation scheme, which allows efficient descriptions of a multiplier from a circuit representation or an abstract specification. An algorithm to detect inequivalence between two arbitrary IBDD multiplier graphs has also been developed. We have also shown how the IBDD representation scheme can be further augmented by employing functional partitioning, leading to efficient representations of most difficult functions discussed in the literature.

Our results show that the specification graphs for all outputs of a 16-bit multiplier can be generated in 4 seconds and the circuit implementation was verified against its abstract specification in less than 20 minutes. The IBDD graph for the 16^{th} output of this multiplier can be represented in 3,892 nodes, contrasted with a million nodes required by any OBDD representation. Similar results exist for other difficult functions. For an 127 input Hidden Weighted Bit (HWB) function, IBDDs required only an 8,257 node graph. An OBDD representation for the same function could require more than 239 Billion nodes.

FERRARI: A Tool for the Validation of System Dependability Properties

A major step toward the development of fault-tolerant computer systems is the validation of the dependability properties of these systems. Fault/error injection has been recognized as a powerful approach to validate the fault tolerance mechanisms of a system and to obtain statistics on parameters such as coverages and latencies. We have developed FERRARI (Fault and ERRor Automatic Real-time Injector), a fault injector implemented in software, to aid in the evaluation of large and complex fault-tolerant systems. The primary features of FERRARI are: 1) it is software implemented; 2) it allows controllability over the location (inside the source code) of fault or error, the type and duration of fault or error injected, and time of injection during the execution of the application; 3) it has the ability to emulate hardware faults as well as control flow errors through software emulation (bus errors, memory errors, and processor control line errors can be injected); 4) it can measure error detection latency in terms of the number of instruction execution cycles as well as elapsed micro seconds; 5) it is able to locate an error that is either detected by an error detection technique or has led to a system failure; 6) it is automatic and is user transparent. The main contribution of FERRARI is its ability to inject both permanent and transient faults. The increased concern with transient faults has led system designers to incorporate concurrent error detection capabilities into digital systems. In order to test the effectiveness of these error detection as well as correction mechanisms, it is imperative that a fault and error injector be able to inject transient faults. The current version of FERRARI has been implemented on SPARC workstations in an X-window environment.

Model	Description
1	address line error resulting in executing a different instruction
2	address line error resulting in executing two instructions
3	address line error when a data operand is fetched
4	data line error when an operand is stored
5	data line error when an opcode is fetched
6	data line error when an operand is loaded
7	data line error when an operand is stored
8	errors in condition code flags

Table 1: Selected Transient Error Models

FERRARI is composed of four modules, the manager, the initializer and activator, the fault and error injector, and the collector and analyzer. The manager module controls the flow of information and commands among the other three modules of FERRARI which are described below.

The fault and error classes supported by FERRARI are: 1) hardware, 2) control flow, and 3) user defined fault and error classes. The hardware models are implemented to emulate real hardware faults and errors. These models include memory faults and errors, external bus faults and errors, and faults in opcode decoding circuitry, instruction prefetch circuitry, program control unit, data registers, ALU, and multiple faults on external buses. Either permanent faults or transient errors can be injected.

Models in the control flow fault and error class are a subset of the hardware models presented earlier, and were added to facilitate evaluation of control flow error detection techniques. Errors in this class emulate both control bit errors and control flow errors. A control bit error is a bit error in the instruction which results in executing a different instruction. On the other hand, a control flow error causes an incorrect execution sequence. A control flow error is injected by changing the value of the program counter, changing the target address of a branch instruction, or the execution of a trap instruction. In the user defined class, the user specifies a location in the source code as well as a condition for fault injection. A condition is an evaluation of a Boolean expression in the user program such as a loop index or a data variable. A fault in this class is injected only when the execution of the program reaches the selected location and the condition is satisfied.

FERRARI supports the injection of both permanent faults and transient errors. The mechanisms for fault and error injection are identical in both cases. The only difference is in the duration of the injected fault and error. For transient error injection, the duration is defined to be one machine cycle. On the other hand, the duration of permanent faults may be several cycles, or may span the entire execution interval of the application. Some of the transient error models supported in FERRARI are listed in Table 1. These were used in our experiments.

Transient errors are injected as follows. When the execution reaches a specified address, the program is trapped. A selected error is injected and the current instruction is executed. The injected error is then removed and the program is allowed to resume execution. The

reason for this procedure is to avoid injecting the error more than once if the selected address was in a loop. Of course, if the single execution of the instruction under the error resulted in a change of internal state, this erroneous state would remain, and may cause other execution errors subsequently.

Results for every run are logged. Each run records the location of the fault or error (virtual address), the affected bit, and the affected register, if any. The monitor appends status flags for every run indicating whether the fault was dormant (did not lead to a failure during the lifetime of program execution or was overwritten), the resulting error was detected, or it did lead to a failure. In addition, the monitor logs the identity of the error detection mechanism and the error detection latency if the error was detected. The collection and analysis module collects these results along with the associated status flags and determines the count for each incident. At the end of the experiment, the module determines the count and percentages (with respect to coverage, latency, type of error detection mechanism) for each run.

Over 600,000 experiments were conducted to demonstrate the capabilities of FERRARI as well as to study the behavior of target systems under faulty conditions. In addition, a variety of faults and errors were injected to measure the effectiveness of error detection and correction techniques that were built into several applications. The experiments were performed on SUN4 SPARC workstations running SUNOS 4.1.1.

Results of the experiments show that the coverages obtained were very dependent on the fault and error models chosen. In particular, coverages of transient errors were shown to be quite different from errors due to permanent faults, or when faults are injected into the memory image. Analysis of the cause of undetected errors was shown to help in the design of better error detection mechanisms. FERRARI can be easily ported to other architectures and operating systems since it has been designed in a modular fashion, and is implemented in C++.

Publications

Journal Papers

V. S. S. Nair and J. A. Abraham, "Real-Number Codes for Fault-Tolerant Matrix Operations on Processor Arrays," *IEEE Transactions on Computers (Special Issue on Fault-Tolerant Computing)*, vol. 39, April 1990, pp. 426–435.

V. S. S. Nair, Y. Hoskote and J. A. Abraham, "Probabilistic Evaluation of On-Line Checks in Fault-Tolerant Multiprocessor Systems," *IEEE Transactions on Computers (Special Issue on Fault-Tolerant Computing)*, vol. 41, May 1992, to appear.

Invited Papers

V. R. Shamapant, J. A. Abraham and D. G. Saab, "Accurate Characterization of Error Propagation in a Highly Parallel Architecture," *Proc. SPIE Annual Symposium*, vol. 1348, San Diego, California, July 1990, pp. 518–527.

V. S. S. Nair and J. A. Abraham, "A Probabilistic Model for the Evaluation of Fault-Tolerant

Multiprocessor Systems using Concurrent Error Detection," *Proc. SPIE Annual Symposium*, vol. 1348, San Diego, California, July 1990, pp. 545–555.

Refereed Conference Papers

V. S. S. Nair and J. A. Abraham, "Hierarchical Design and Analysis of Fault-Tolerant Multiprocessor Systems using Concurrent Error Detection," *Proc. 20th Int'l Symp. on Fault-Tolerant Computing*, Newcastle, U.K., June 1990, pp. 130–137.

J. Jain, M. Abadir, J. Bitner, D. S. Fussell and J. A. Abraham, "IBDDs: An Efficient Function Representation For Digital Circuits," *The European Conference on Design Automation*, Brussels, Belgium, March 16-19, 1992, pp. 440-446.

J. Jain, J. Bitner, J. A. Abraham and D. S. Fussell, "Functional Partitioning for Verification and Related Problems," *MIT/Brown Symposium on VLSI*, Providence, RI, March 25-27, 1992, pp. 210-226.

G. A. Kanawati, N. A. Kanawati and J. A. Abraham, "FERRARI: A Tool for the Validation of System Dependability Properties," *Proc. 22nd Int'l Symp. on Fault-Tolerant Computing*, Boston, MA, July 1992, to appear.

Submitted Journal Papers

V. S. S. Nair, J. A. Abraham and P. Banerjee, "Analysis of fault-tolerant multiprocessor systems using concurrent fault diagnosis," *IEEE Transactions on Computers*, under review.

V. S. S. Nair and J. A. Abraham, "Hierarchical design and analysis of multiprocessor systems using concurrent error detection," *IEEE Transactions on Computers*, under review.